

Advantages and limitations of using a server cluster for Server Appliances (specifically, X - Web Form Manager by Avain Technologies Oy)

Alexei Mikhailov

Authors Alexei Mikhailov	Group
The title of your thesis Advantages and limitations of using a server cluster for Server Appliances (specifically, X - Web Form Manager by Avain Technologies Oy)	Number of pages and appendices 33
Supervisors Markku Somerkivi	
<p>This thesis describes some general concepts and logic behind server clusters, as well as gives a few examples on technologies available as of 2009.</p> <p>It also depicts a certain problems that can occur when using clustering environment in a certain situations. As an example, a test of X-Web Form Manager application by Avain Technologies is presented.</p> <p>The objective of this paper was to find out the basics of clustering technologies and to try out X-WFM in a new environment.</p>	
Key words Server clusters, clustering, web applications, javascript, java, xml, jetty, cocoon	

Table of Contents

1	Introduction.....	2
2	Theoretical part: Clusters.....	3
2.1	What is clustering.....	3
2.2	Need of high availability.....	4
2.3	Types of HA clusters.....	6
2.3.1	What is virtual ip.....	6
2.3.2	HA IP clusters.....	7
2.3.3	HA application clusters.....	8
2.4	Available solutions: Unix-based systems.....	11
2.4.1	Linux HA.....	11
2.4.2	OpenAIS.....	11
2.4.3	Red Hat Cluster Suite.....	12
2.4.4	Veritas Cluster Server.....	12
2.5	Available solutions: Windows.....	13
3	Theoretical Part: X – Web Form Manager.....	14
3.1	Basic structure of X – WFM.....	15
3.2	Possible problems with using X – WFM in clustering environment.....	18
4	Empirical part.....	20
4.1	Stress testing web servers with JMeter.....	20
4.2	Setup.....	22
4.3	Expected results.....	24
4.4	Test case.....	24
4.5	Results of stress-testing X – WFM on Windows.....	28
4.6	Result assertion.....	29
4.7	Possible solutions.....	30
5	Summary and conclusion.....	32
6	Bibliography.....	33

1 Introduction

Ever since first web-servers emerged, there was a problem of systems failing due to heavy load on the server, and thus whole service being unavailable during the server down-time. Clustering several servers arise become one of the most popular ways to battle this problem. Nowadays, high availability is one of the biggest need for any online service that should be available 24 hours a day. So the support of HA is among the key aspects of success for any big application.

Apart from availability, all HA-systems provide greater overall performance and capacity, or some combination of these three. One of the main goals of HA-systems is to minimize system down-time due to malfunction of either software or hardware components. However, there are several different ways of implementing HA system, which provide a bit different functionality and are priced differently.

In this paper, I'll describe the general structure of high-availability clusters, and show the difference between cluster implementation and single server implementation of a web application using the X – Web Form Manager by Avain Technologies as an example. Several examples of ways to stress test web application using JMeter will be shown, and general idea behind this kind of testing depicted.

Structure of X–WFM will be shown as described as well, possible drawbacks of using it in a clustered environment discussed, and possible solutions provided.

2 Theoretical part: Clusters

2.1 What is clustering

As noted in introduction, computer cluster is a system, consisting of two or more server nodes, running the same application. In case one node fails, requests that were supposed to be processed by that node are redirected to other nodes, by that ensuring that the application is still available. Clustering technologies also make it possible for multiple servers to work in unison, making, the whole system that runs on cluster appear to the outer world as a single computing environment [*HP-UX System Administrators Guide (Version 3), Chapter 2*]. Even though technically each server is running its own operating system and software, they work together as if they were one, ensuring the minimal down-time of the system.

As you might have noticed, this comes hand-in-hand with general idea of high availability systems, which states that everything should be duplicated [*High Availability Fundamentals*]. Modern servers have lots of components and features that help them run day and night without any need to shut them down for any maintenance. Quite many implementations also allow hot-swap of any components (including critical ones, like processors, network interfaces, even power supplies) in no time. However, the weak point remains – server is one single machine, physically located in one certain geographical place. What if fire happens in the server room? Or an accident leads to a water flood? Regardless of how well the server is protected from any other outside hazard, it still stays as a weak link in a chain from company providing service to end user in case it is the only server used.

Now if we zoom out to a higher abstraction perspective of a server application – servers are one of the components of it. Thus, the most logical way to ensure high availability of our application would be to duplicate them, and voila – we got a cluster of servers.

2.2 Need of high availability

Failures are generally impossible to eliminate completely, regardless of how well the system is designed or how durable and bullet-proof it's components. Nevertheless, it is possible to manage the failures, thus minimising the impact on the system [*HP-UX System Administrators Guide (Version 3)*] .

There are two different types of down-time: planned and unplanned [*HP-UX System Administrators Guide (Version 3)*]. Making each of these as minimal as possible requires different approaches. Planned down-time includes time for system updates and maintenance which requires shut down of the main application. Those are getting more and more rare nowadays because of both new engineering solutions to providing possibility of hot swap of critical server components, as well as because of new software features and improvements (like recent Linux kernel (the heart of any Linux distribution) update, making it possible to update it without any need for restart afterwards).



Illustration 1: Silicon Graphics Cluster (photo from Wikipedia)

Unplanned down-time includes all other system unavailability reasons [*HP-UX System Administrators Guide (Version 3)*] . That might be a software bug (for example one causing huge memory leaks and slowing down the whole system), network issues and problems (too much network connections to a single server), or physical accident (which is

quite unlikely to happen in modern server rooms, but still); everything that is rather hard to predict, in other words. Even though HA systems are mostly associated with minimizing unplanned down-time, they also proved to be useful during planned down-time.

Because term “clustering” itself is so broad, often other terms are used to describe different cluster implementations (failover, load balancing, parallel and grid computing). Grid computing (like [SETI@home](#), one of the largest distributed grid, using three million home computers all over the world to analyse data from the Arecibo Observatory radio telescope) also uses clusters, but is focused mostly on throughput rather than minimal down-time. However, HA clustering solutions mostly concentrate on enhanced availability for a single service or application, which is the scope of this research.

2.3 Types of HA clusters

According to *HP-UX System Administrator's Guide* HA clusters can be subdivided into two common types: HA IP clusters and HA application clusters. However before discussing those two, one common term should be described – that is **virtual ip**, since both types of clustering use that one extensively.

2.3.1 What is virtual ip

A typical IP address resolves to a single server. Now, let's consider the situation when we have a set of servers that have the same content, but unique (and sometimes private) IP addresses, behind a hardware switch. In this situation, only hardware switch is assigned a public IP address; it receives all incoming public requests and then redirects them to a host in the cluster, basing on load. Thus, IP address of a switch is actually a virtual IP that represents a clustered application servers. The responses can be returned directly from application servers, or the same way they came (via the switch) [*IBM iSeries Information Center, Version 5 Release 3*].

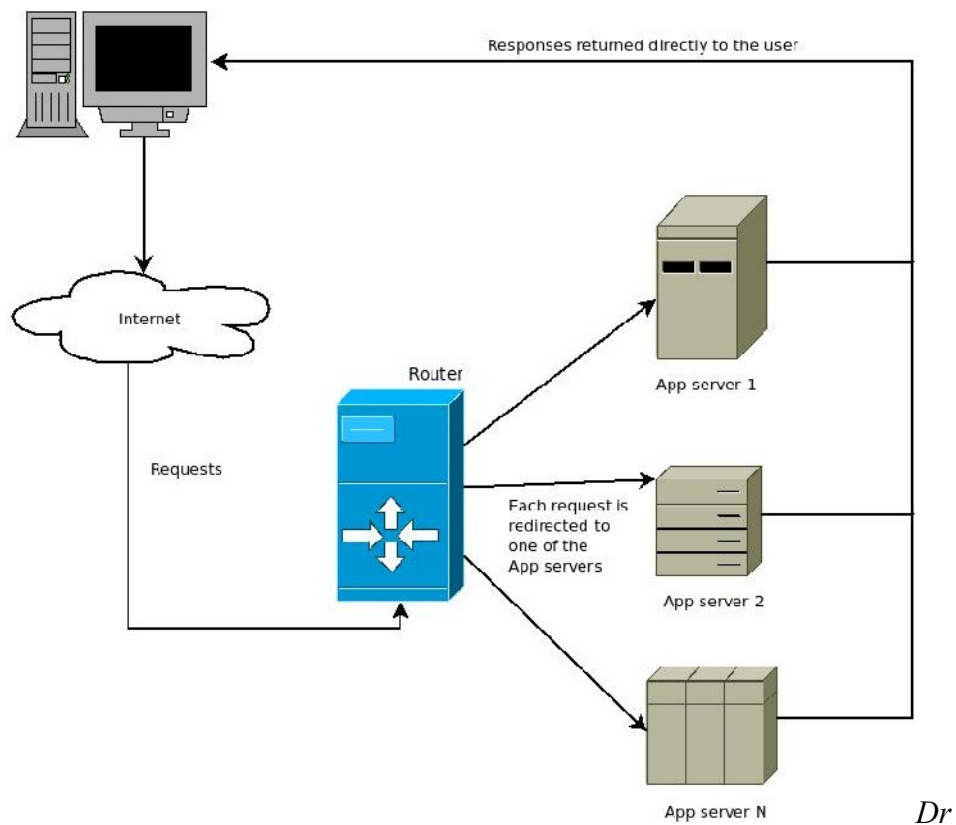


Figure 1: An example of network structure with virtual IP

2.3.2 HA IP clusters

HA IP clusters are used to ensure availability for network access points (which are typically IP addresses used to access network services). They are usually represented by the virtual IP address (or addresses) clients use to access the clustered services [*HP-UX System Administrator's Guide*].

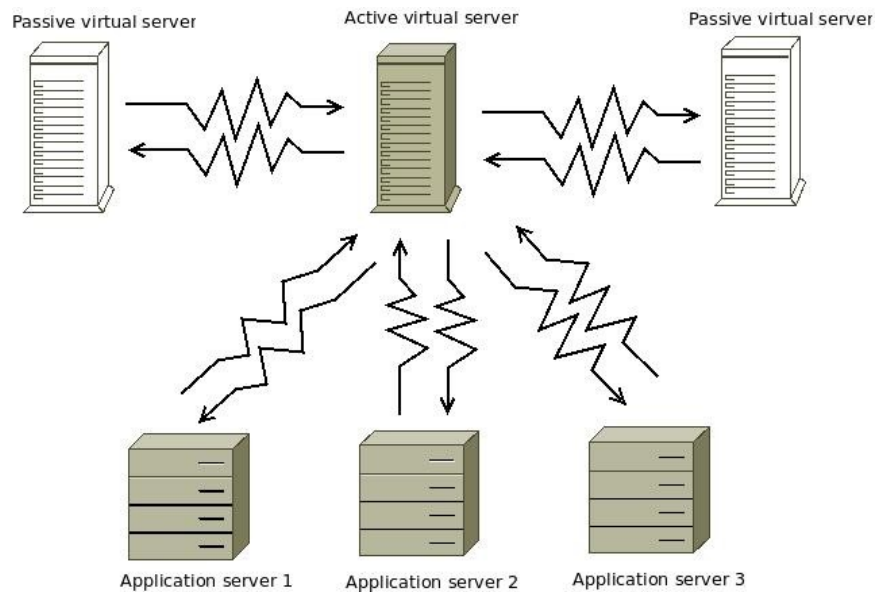


Illustration 2: Monitoring health of servers with heartbeats

This is usually implemented with a certain mechanism (Linux Virtual Server (LVS) in case of Linux HA, for example), which provides virtual IP support and can also load-balance applications (in case applications data is replicated on a pool of application servers).

Basically, virtual server represents application servers to network clients as if they were a single system. They are kept unaware of the physical IP addresses used by virtual server, as well as of physical addresses used by application servers; the clients access only virtual IP address managed by virtual server [*IBM iSeries Information Center, Version 5 Release 3*].

Virtual server is responsible for routing requests to application servers. High availability here is achieved by having multiple destinations that can process request – even if one of application servers goes down, one or more will be capable to process re-

quest. Same goes for the IP cluster itself – if one of the servers comes out, there are always other servers that can receive request.

High availability is also ensured by monitoring servers health. This monitoring is usually performed with so-called heartbeat mechanism. Heartbeat packets are constantly being sent between cluster nodes at regular time interval (usually one every couple of seconds). If a heartbeat is not received in a certain period of time, the application server is assumed to be down, and requests are being redirected from it to another server until it comes up again [*Basic structure of Linux-HA*].

Even though HA IP is not really an application balancer, it can be used quite effectively for load-balancing static Web and File Transfer Protocol (FTP) servers, as well as video streaming servers.

2.3.3 HA application clusters

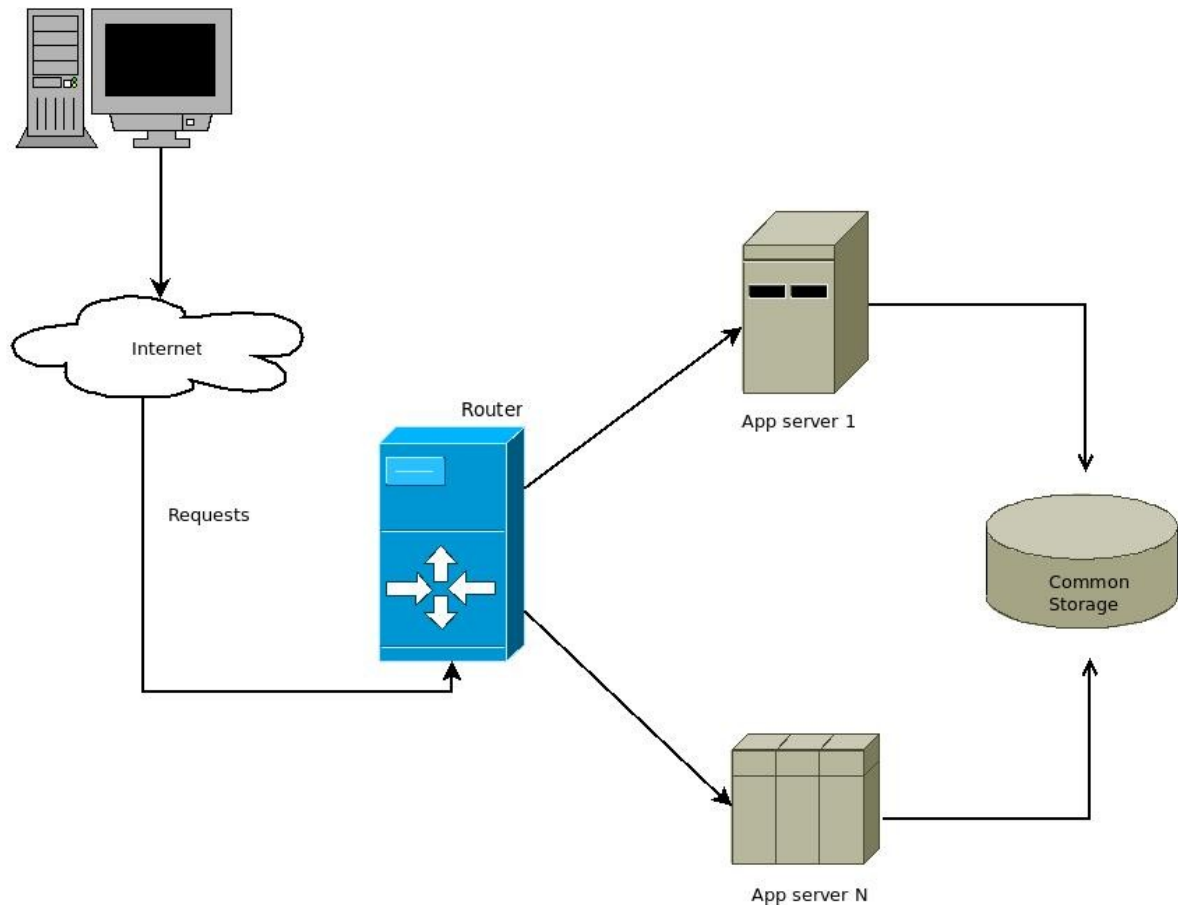
Transactional application is an application where every operation is a transaction between application and external service. A database server would be a good example of a transactional application. These usually use HA application clusters to ensure high availability, which is achieved through monitoring continuously the health of an application in question and the resources the applications depends on for normal operation, including the server the application is running on. If application or server starts to malfunction, the cluster will restart (*failover*) it on one of the remaining servers, until failing node is back to normal operation state [*HP-UX System Administrator's Guide*].

Resources used by application (IP addresses, software modules, disks) may be also overtaken by other nodes, depending on the implementation taken in use. Generally, there are three approaches to ensure the availability of application data or storage to reminding servers: “mirroring”, “shared nothing” and “shared everything” [*HP-UX System Administrator's Guide*].

Mirroring involves constant copying of data between separate data storages of application servers. This one ensures high recovery rate (since each node will have a copy of most data), but will result in big network and system overhead due to those data synchronization operations. Also a data loss might occur in case of failover.

“Shared nothing” and “shared everything” are in a way opposite to each other. In “shared nothing”, each application server has its own storage, meaning that in case there is some temporary files that are stored on hard-drive (like counters), while with

“shared everything” all nodes share almost all available resources (except for CPU), which involves quite complex systems that can lock certain parts of memory or hard drive for certain node. An intermediate version would be “shared disk”, when all nodes share the same storage system. This is quite common in case of fail-over clusters – since usually only one instance of application is running, and when it fails, the backup cluster gains rights to the storage system.



Drawing 2: Active-active system with shared storage

HA application clusters also have their own implementation of active/passive and active/active concepts. A common approach to achieve high availability is the passive standby mode, when one server acts as the primary node, and having a secondary server as a backup in case of failure [*TCP/IP for dummies, 6th edition*]. The secondary server doesn't do any data processing, but instead just stands by to take over if the primary server fails, which enables maximum resources to be available to the application during failure.

On the other hand, this is quite an expensive way to implement HA cluster, since it requires twice the amount of hardware to be purchased without any performance increase for the application.

Thus, for most of the applications, active/active configurations can be more effective. These one also consist of two servers; however each server performs useful processes, while retaining the ability to take over for another server in case of emergency. On the other hand, implementation of active/active configuration will also result in increased design complexity, as well as the potential performance issues in case one or several servers go out of order [*TCP/IP for dummies, 6th edition*] .

2.4 Available solutions: Unix-based systems

2.4.1 Linux HA



One of the most known existing HA cluster solution is, among the others, Linux HA. As projects site says, *”The Linux-HA project is a widely used and important component in many interesting High Availability*

solutions, and ranks as among the best HA software packages for any platform. We estimate that we currently have more than thirty thousand installations up in mission-critical uses in the real world since 1999. Interest in this project continues to grow.”

[Linux-ha.org main page]

Linux HA supports most of the features required for efficient clustering, like load balancing (with Linux Virtual Server), load distribution, data replication, time-based events, no fixed maximum number of nodes, etc. It also includes GUI for configuring, controlling and monitoring resources and nodes.

It works with Linux, FreeBSD and OpenBSD, Solaris and Mac OS X. It is also open source, with all its benefits and drawbacks. It doesn't have any specific system requirements, and supports lots of platforms and architectures, including ia32, ia64, x86_64, pSeries, zSeries mainframes.

Full feature list can be found at project pages:

<http://linux-ha.org/FactSheetv2>

Basic architecture is described there as well:

<http://www.linux-ha.org/BasicArchitecture>

2.4.2 OpenAIS

Another linux-based implementation of a clustered servers would be OpenAIS. It is being ship as a part of a default package together with many popular linux distributions, including Debian (starting from latest (2009) stable release ”Lenny”), Ubuntu, Fedora, Red Hat and SUSE. OpenAIS mostly concentrates on providing high-availability features, but not so much of a load balancing features.

To quote the description from official web site, "the *OpenAIS Standards Based Cluster Framework* <...> is a software API and policies which are used to develop applications that maintain service during faults. Restarting and failover of applications is also provided for those deploying applications which may not be modified. The *OpenAIS* software is built to operate on the *Corosync Cluster Engine* which allows any third party to implement plugin cluster services using the infrastructure provided."

2.4.3 Red Hat Cluster Suite

Red Hat Cluster Suite provides both application failover cluster features (clusters consisting of several server nodes for applications failover), as well as IP Load Balancing (load balance incoming IP networks).

The key benefit for big companies in using Red Hat solution is, of course, a possibility to buy a support license, so the company is in a way protected from possible losses in case the system malfunctions. This is more of an exception than a rule in Linux world, however.



2.4.4 Veritas Cluster Server

Veritas Cluster Server is developed by Symantec, and provides most of a required features of high availability clusters to decrease both planned and unplanned downtime. One of the interesting features of Veritas is that it supports heterogeneous physical and virtual operating system platforms with out-of-the-box solutions for all major database, application, and storage vendors – that, naturally, makes it quite attractive for business with important mission critical application servers.

2.5 Available solutions: Windows



For Windows operating system, HA solutions are mostly presented by Microsoft itself, with either Windows Compute Cluster Server 2003 or Windows HPC 2008. Another option would be everRun HA or everRun FT by Marathon Technologies, which both provide active/passive or active/active clustering solutions respectively. These ones, however, are based on virtualisation technologies rather than hardware solution, so its out of the scope of current research. Nevertheless, seems like with every new Windows Server version support of virtualisation (and, specifically, Marathon products) is increased.

Both Microsoft solutions are based on Windows Server edition (2003 and 2008 respectively), adding a number of features like better failover support and virtualisation services. There is also quite big amount of graphical tools available to administrators for monitoring servers health, performing maintenance, and other operations.

The latest release of Windows Server requires quite an advanced hardware (even in case of the simplest “Foundation” version), and works only with 64-bit architecture.

It also provides Failover API and two APIs: the Failover Cluster API and the Cluster Automation Server. These Failover Cluster APIs let programmers develop management tools and high-availability resources for failover clusters. (*MSDN, [http://msdn.microsoft.com/en-us/library/cc296100\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/cc296100(VS.85).aspx) – accessed 20.11.2009*)

3 Theoretical Part: X – Web Form Manager

X – Web Form Manager is an integrated product developed by Avain Technologies Oy. It implements an idea of paperless document workflow, providing its users with possibility to submit information through web-forms and process it through a customizable workflow, where each phase can be secured by digitally signing it. Access rights to the forms, their phases and certain fields can be also configured [*X-WFM Documentation*].

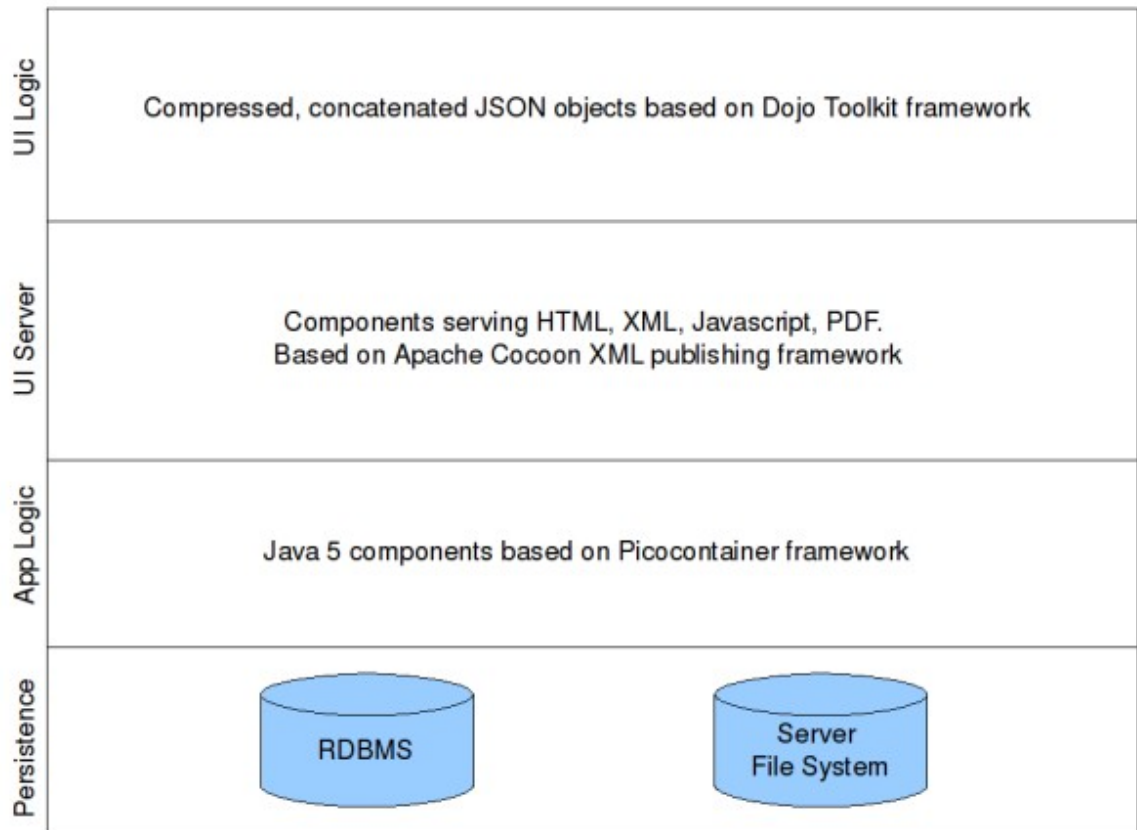
X – WFM suite consists of X – WFM itself, X – DSS (Digital Signature Suite), which makes it possible to digitally sign X – WFM forms, and X – Archive, which enables system to store information according to the archiving laws. This makes the system especially useful for the organizations that deal with sensitive information (like hospitals) – there is no need any more to store archived information on paper, since its digital copy already has legal force.

The whole form processing system is based on a workflow model, thus the forms are processed much as a traditional paper forms are, only with all advantages of an online system.

The system is based on the Java 5 platform, and uses SQL relational database management systems to store information. It is highly portable to a variety of operating system and application server platforms, including IBM WebSphere, Oracle Internet Application Server, Microsoft IIS on Windows, and Apache 2 with Jetty 6 on Linux.

3.1 Basic structure of X – WFM

X – WFM is a four-tier structure.



Drawing 3: X – WFM basic structure (c) Petteri Sulonen

User interface logic is executed on the browser-side of the application, and includes X – WFM form viewer and user desktop, as well as administration tools and X – DSS Signature Widget (which will require client-side Mozilla plugin or Active-X control for client certificate signatures).

It is represented mostly by concatenated and compressed Javascript files, which extend Dojo Toolkit Development framework, and CSS styled HTML pages, which are served by the server.

User interface server tier is a connection layer between UI logic and Application logic tiers. It is based on Apache Cocoon framework, and mostly serves HTML pages to UI tier by transforming static or externally received XML files. It is also used for internation-

alization purposes, as well as rendering PDF's, generating cache, or any other task that would require any level of transformations of input XML file.

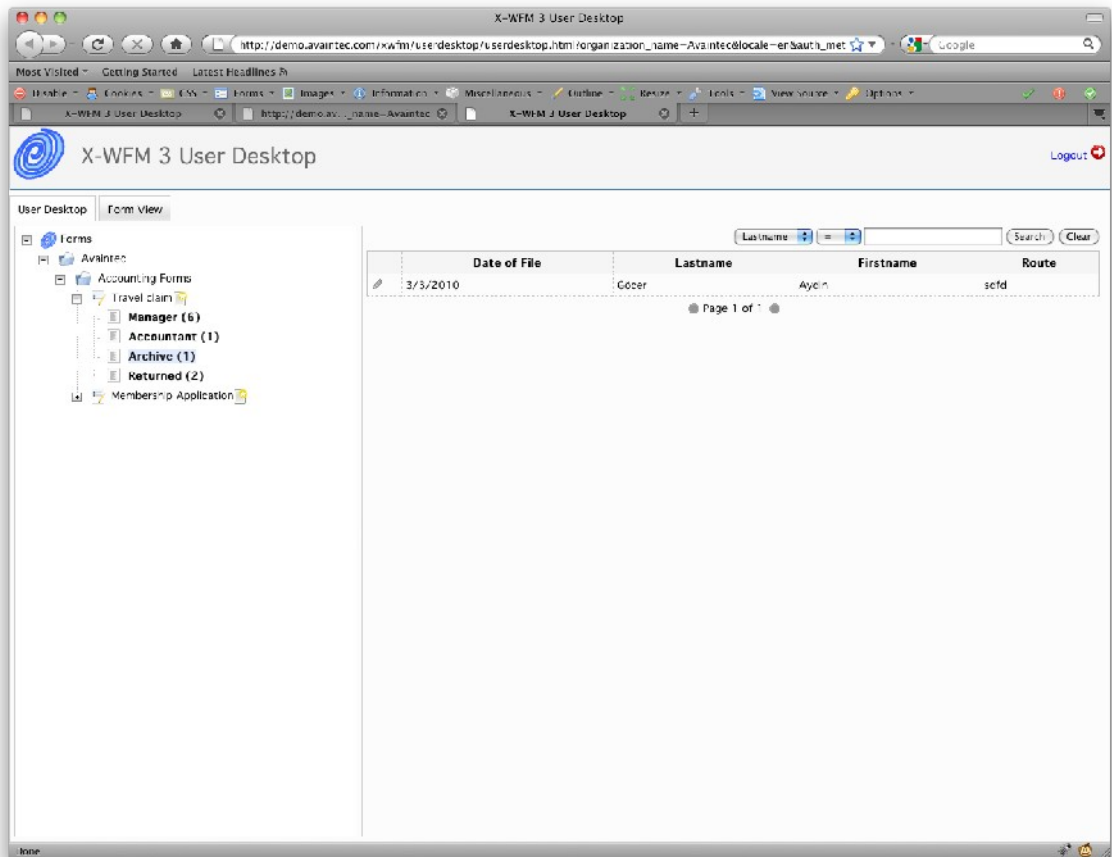


Illustration 3: X-WFM 3 User Desktop

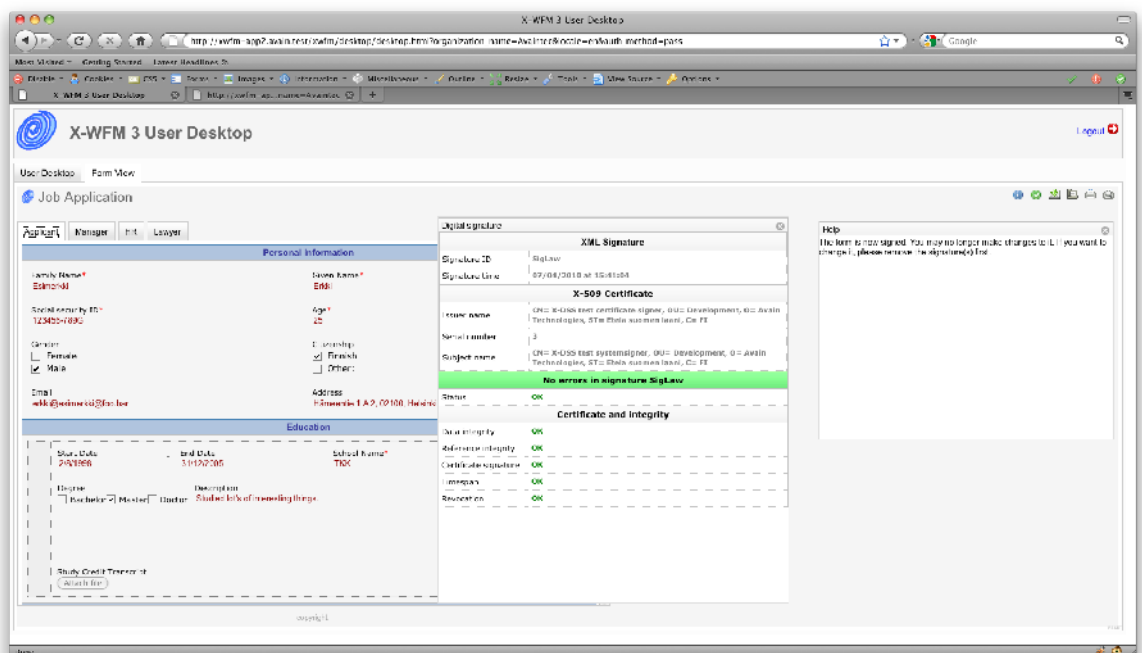


Illustration 4: X-WFM 3 View with signature widget.

The Application tier is based on Java 5 and uses Picocontainer framework for dependency injection (in short that is a way of encompassing Inversion of Control main principle to keep dependency resolution, configuration and lifecycle of a component out of component's concern).

It does all the internal processes in X – WFM, be it connecting to Persistence layer, recovering from errors, logging or reporting. The structure of components in Application layer is module-based, so it is considerably easy to improve or substitute one module without the need of rewriting the whole application.

Finally, Persistence tier consists of relational database management system and server file system. Forms, their related information and user rights information are stored in database, while logs and counters are stored on the file system [*X-WFM Documentation*].

3.2 Possible problems with using X – WFM in clustering environment

The system is considerably fast even when operating on only one server – even though there is a somewhat complex processing involved in extracting information from the database (mostly due to implementation of different legal requirements related to storing and archiving information) during transactions – even during heavy load the wait time for user will rarely be more than 30 seconds. The benefit of using two parallel servers, each of which will handle half calculations of each request won't be big. Even worse, this might introduce certain slowness or inconvenience, since system originally wasn't optimized to work in a clustered environment.

Same goes for the fail-over cluster scenario: X – WFM, running on different servers, would have to have a certain shared storage for its temporary files (mostly transaction information is stored there for back tracking and queueing requests). Hence there is a problem in case there is no such storage present. For example, X – WFM stores it's own internal counters in a separate files on the server hard-disk. These files contain (among other things) instance id's for each separate record in X – WFM. In case of more than two servers, this might cause problems in case the servers hard drives are not synced (for example failover server will try to create new instance with id 155, even though main server already used id 155).

This, however, is already partly solved by having a separate prefix for instances ids. Thus, each server in a cluster can have it's own index, so it can be guaranteed

that instance id's of one server won't ever duplicate instance id's of other servers (so in example before, instance id's that will go to requests and/or database will be 1155 and 2155).

On the other hand, since we are talking about active-passive system here, it should be relatively easy to setup a shared storage than in case of active-active system – there would be no need to lock files for editing when one of the systems reading from/writing to the file, for example. Still, system administrators are not always willing to create such a shared storage, preferring the situation when each application server has its own hard drive with its own copy of all resources, temp files and other.

Also, it is not quite clear, how will different modules of X – WFM will behave when shut down in middle of processing. Theoretically, they should be able to wait until processing is finished, and then stop properly, but you can never know for sure.

Otherwise, there should be no problems.

4 Empirical part

4.1 Stress testing web servers with JMeter

Apache JMeter is an open source "pure Java" application for stress testing the web applications. It is designed to be useful not only for testing web servers and their components (jsp servlets, or CGI scripts), but also an FTP servers and even databases (using a JDBC driver). It also supports authorization mechanisms of virtual users and user sessions, so a heavy load situation when big amount of information is transferred between application servers and outside world can be simulated.

Alongside with stress testing, Jmeter also provides a certain level of regression testing by asserting the results returned by application.

Since it is a Java application, JMeter requires a JRE/JDK pre-installed on the computer you want to run your tests from. It also provides necessary tools to create a test plan, which you can save locally and run later on.

Since stress testing usually has to be performed on the actual production servers, you can also define a timeout for the test plan to start (for example run the test for 50 000 threads starting from 1:00 in the night till 5:00 in the morning). Test plans are stored as an xml files. From user interface they will look something similar Illustration 4.

A usual test plan consists of one or more thread groups, logic controllers, sample generating controllers and, listeners, timers, assertions, and configuration elements (*Apache JMeter user manual, "Building a test case"*). In my test, I've used mostly thread groups and listeners to be able to monitor the results.

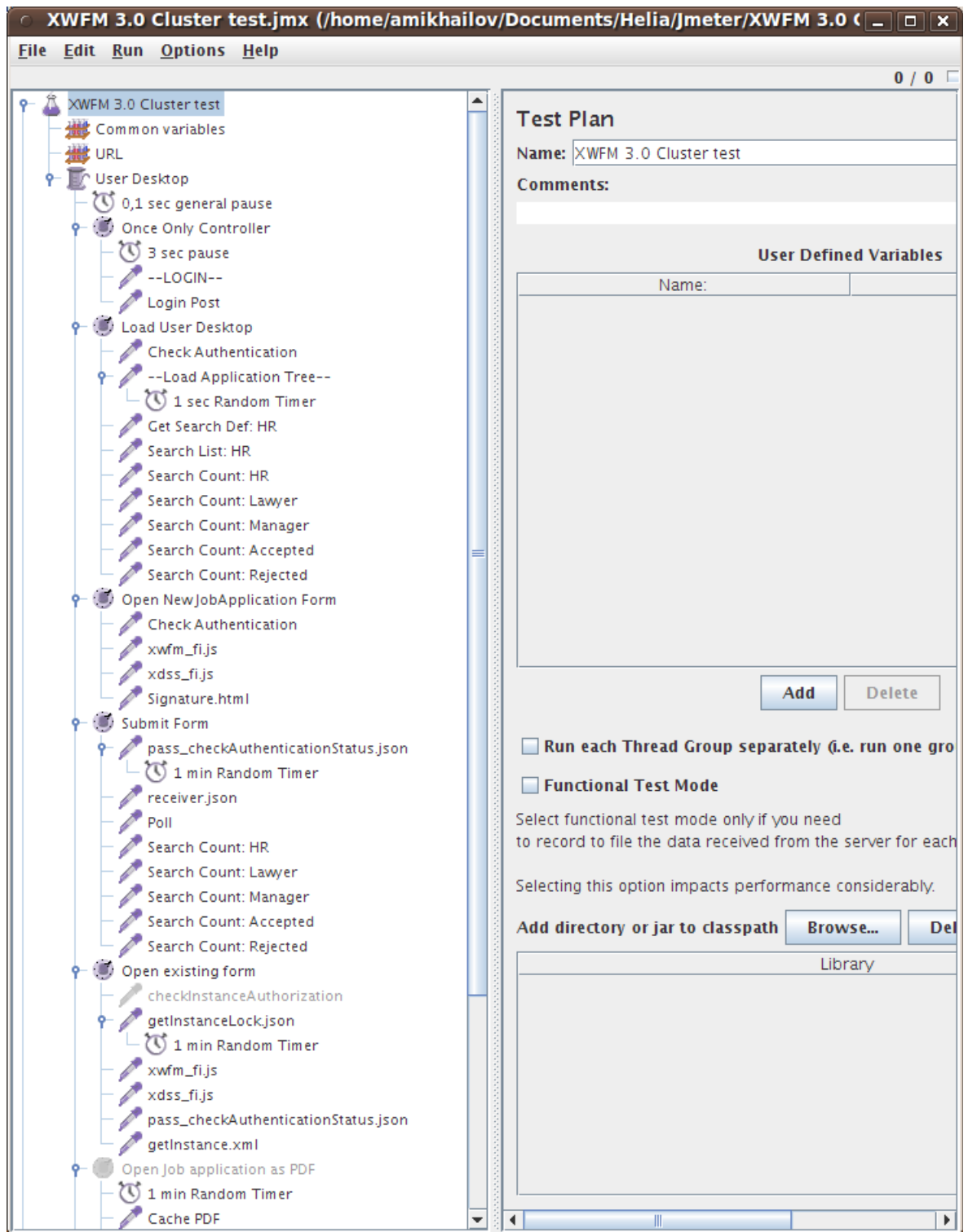


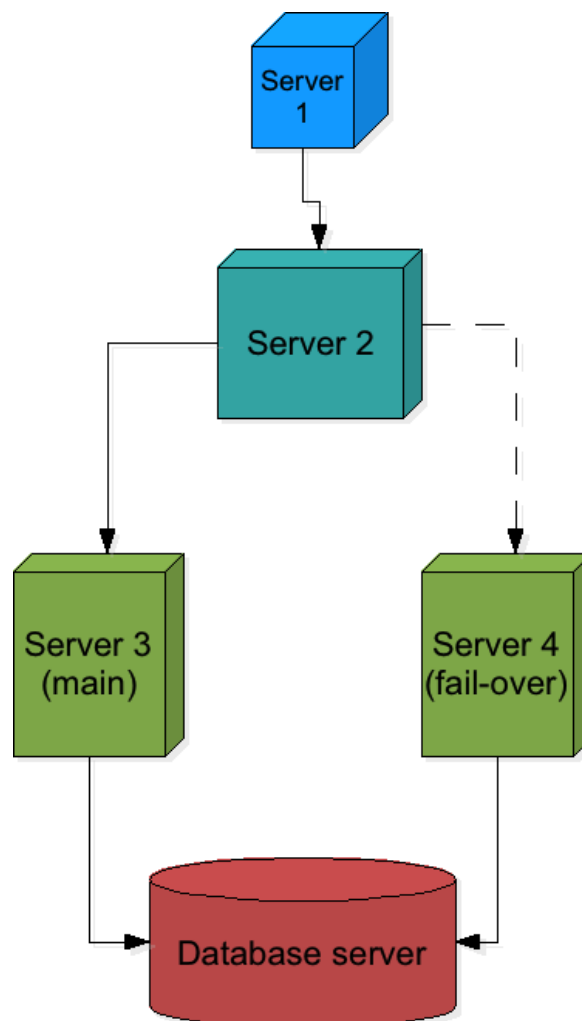
Illustration 5: An example of JMeter test plan.

However, I wasn't really that interested in results parsed by JMeter. The only reason I've used it is for its ability to create heavy load on the server.

4.2 Setup

The following server configuration was used for the stress testing:

- Servers 1, 2 and 3 are identical machines, running Microsoft Windows 2003 Standard edition with Service Pack 2 and security updates by November 2009. They have the same Java JRE 1.5.0_16. The X-WFM installations are also identical (version 3.1.0, built on September 23, 2009)
- JMeter tests were run from a usual notebook.
- Server 1 is the main node, which redirects all the requests to server 2
- In case Server 2 fails – all the requests were redirected to server 3.
- Finally, both Server 2 and Server 3 are using the same MSSQL database server



Drawing 4: Test environment

Both server 2 and server 3 have the same version of X – WFM installed, and altogether are identical copies of each other. Server 1 basically works as a switch, and also starts redirecting requests to server 3 in case response from server 2 takes too much time. Server 3 is in a “hot stand-by mode”, meaning it has a running copy of X – WFM, so when it receives a request, it can start processing requests immediately.

4.3 Expected results

Since this was pretty much a first round of testing, it wasn't expected that everything will work right away as it should (in fact this probably would mean that something was incorrectly set up).

No big guesses on what problems might occur weren't made. Apart from the things listed in chapter 3.2, the most likely part to break would still be the form processing in the backend modules, which includes quite a lot of talking between different backend modules and the database server.

4.4 Test case

The test case was rather trivial. Since we wanted to test the worst possible case (X – WFM shuts down due to too much incoming requests, and “hot backup” takes it's place), the first step would be to put a heavy load on the server,. JMeter is quite good at that – I've created a rather simple test plan, where couple of hundreds of users log in to the system, list all the forms, submit a new form several times, and log out. I've also added short timeouts between each user login, so that system won't crush immediately.

Next, we should wait for a while until the system will start queueing instances – that would indicate, that the load is heavy enough, and rate of new instances creation is greater than rate of processing those instances. We can tell that queued instances appeared by monitoring transaction temp storage on the servers hard drive: when new instance's log file will have “queued” state, it will mean that it's time to move to the next stage.

That would be to imitate a server failure. We can do that by manually shutting down Jetty web server. This will cause the application to stop as well (and possibly some part of it will crash, if something goes wrong).

Now, since the main server is down, our router server should start redirecting requests to a backup server. The switch won't be immediate, though (depending on the timeout we specify for the router), so we might see few transactions failing with HTTP-status 500 (Service unavailable) in the meantime.

However, when traffic will finally be re-routed, everything should get back to normal. Then we can stop the JMeter, stop the backup Jetty and start up the original Jetty to see, what will happen to instances that were queued or under processing, but weren't

processed completely. This is quite simple to see from the log, if that one is properly configured (“playing safe” here would be to enable debug level of logging on all modules. However, that will seriously slow down the system).

So, in short:

1. Run JMeter to put a heavy load on the server.
2. Monitor transactions temp folder to find out when instances will start to queue.
3. When 2. happens, stop Jetty
4. Make sure that traffic is properly re-routed to backup Jetty, and transactions are being processed.
5. When 4. happens, stop JMeter and fail-over Jetty.
6. Start original Jetty and check what happens to queued and half-processed instances

The test case looks rather compact in JMeter viewer, but takes about 70 pages as an xml file, so I won't put it here.

Below you can find screenshots for each step.

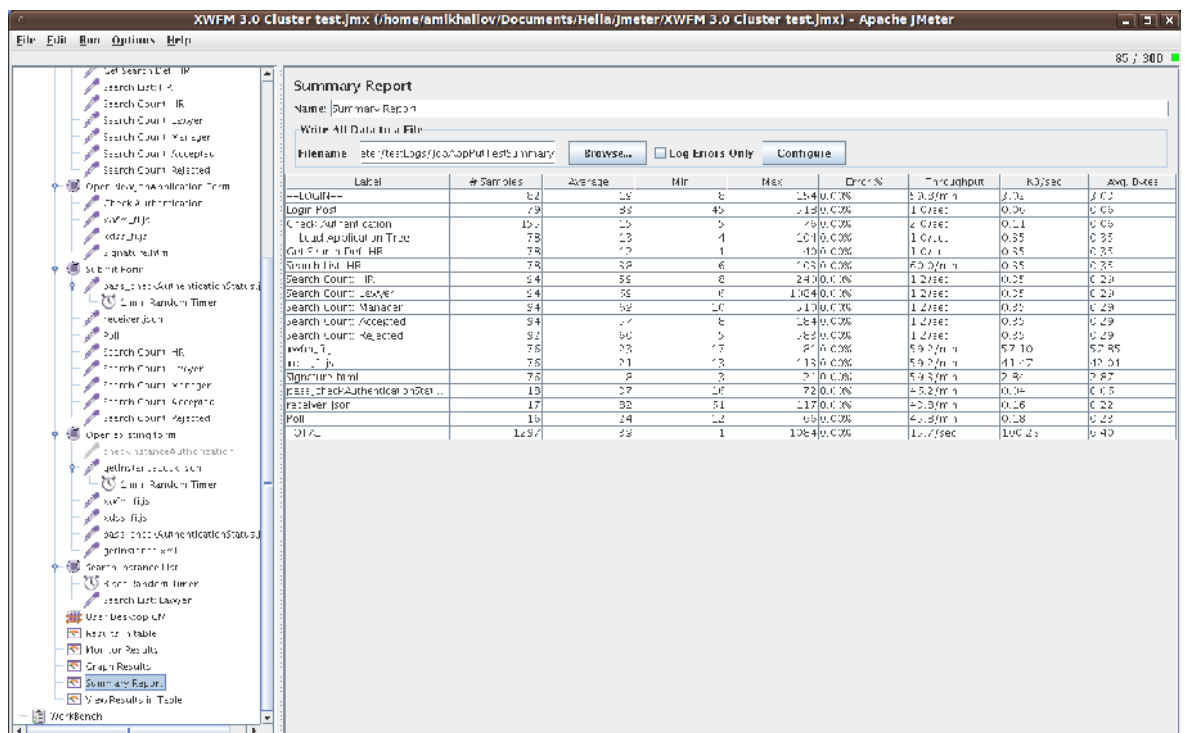
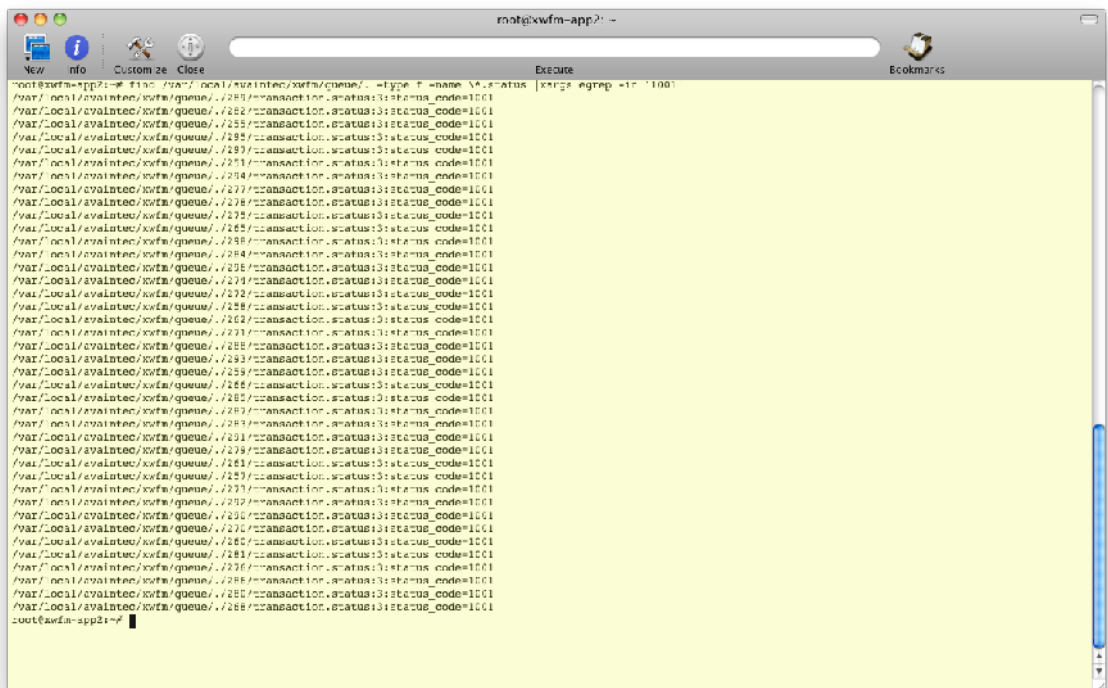


Illustration 6: Jmeter running test



The screenshot shows a terminal window titled 'root@xwfm-app:~'. The terminal prompt is root@xwfm-app:~#. The user enters the command: `find /local/evalntec/xwfm/queue/ -type f -name '*.s' -status | xargs egrep -l '1001'`. The output consists of 26 lines, each representing a file path followed by its status and status code. All files shown have a status of 3 and a status code of 1001, indicating they are queued. The file paths are: `/local/evalntec/xwfm/queue/./.251/transaction.status:3:status_code=1001` through `/local/evalntec/xwfm/queue/./.268/transaction.status:3:status_code=1001`. The terminal prompt returns to root@xwfm-app:~# after the command execution.

Illustration 7: Grep showing instances with 1001 status code, which means that this instance is queued

Sample #	Start Time	Thread Name	Label	Sample Time (ms)	Success	Bytes
11858	13 59:34.443	User Descope LM	Check authorization	72	<input checked="" type="checkbox"/>	430
11859	13 59:34.443	User Descope LM	Search Court: OK	64	<input checked="" type="checkbox"/>	430
11860	13 59:34.447	User Descope LM	Search Court: Manager	252	<input checked="" type="checkbox"/>	59259
11861	13 59:34.444	User Descope LM	Search List: HK	49	<input checked="" type="checkbox"/>	408
11862	13 59:34.444	User Descope LM	Search List: Manager	98	<input checked="" type="checkbox"/>	59259
11863	13 59:34.446	User Descope LM	Search List: Manager	206	<input checked="" type="checkbox"/>	430
11864	13 59:34.455	User Descope LM	Search List: Manager	110	<input checked="" type="checkbox"/>	59259
11865	13 59:34.415	User Descope LM	Search List: Manager	186	<input checked="" type="checkbox"/>	430
11866	13 59:34.442	User Descope LM	Search List: Manager	276	<input checked="" type="checkbox"/>	59259
11867	13 59:34.473	User Descope LM	Search List: Manager	204	<input checked="" type="checkbox"/>	430
11868	13 59:34.447	User Descope LM	Search Court: OK	146	<input checked="" type="checkbox"/>	408
11869	13 59:34.456	User Descope LM	Search Court: Manager	505	<input checked="" type="checkbox"/>	59259
11870	13 59:34.440	User Descope LM	Search Court: Manager	161	<input checked="" type="checkbox"/>	408
11871	13 59:34.475	User Descope LM	Search List: Manager	270	<input checked="" type="checkbox"/>	59259
11872	13 59:34.415	User Descope LM	Search List: Manager	98	<input checked="" type="checkbox"/>	59259
11873	13 59:34.429	User Descope LM	Search List: Manager	194	<input checked="" type="checkbox"/>	59259
11874	13 59:34.433	User Descope LM	Search List: Manager	75	<input checked="" type="checkbox"/>	59259
11875	13 59:34.444	User Descope LM	Search Court: HK	25	<input checked="" type="checkbox"/>	408
11876	13 59:34.456	User Descope LM	Search Court: Manager	11	<input checked="" type="checkbox"/>	2940
11877	13 59:34.458	User Descope LM	Search Court: Manager	17	<input checked="" type="checkbox"/>	408
11878	13 59:34.457	User Descope LM	Search List: Manager	76	<input checked="" type="checkbox"/>	430
11879	13 59:34.458	User Descope LM	Search List: Manager	27	<input checked="" type="checkbox"/>	430
11880	13 59:34.457	User Descope LM	Search List: Manager	58	<input checked="" type="checkbox"/>	430
11881	13 59:34.445	User Descope LM	Search Court: Manager	22	<input checked="" type="checkbox"/>	2940
11882	13 59:34.448	User Descope LM	Search Court: Manager	24	<input checked="" type="checkbox"/>	430
11883	13 59:34.476	User Descope LM	Check authorization	15	<input checked="" type="checkbox"/>	408
11884	13 59:34.470	User Descope LM	Search List: Manager	12	<input checked="" type="checkbox"/>	430
11885	13 59:34.475	User Descope LM	Search List: Manager	48	<input checked="" type="checkbox"/>	2940
11886	13 59:34.479	User Descope LM	Search List: Manager	27	<input checked="" type="checkbox"/>	408
11887	13 59:34.475	User Descope LM	Search List: Manager	80	<input checked="" type="checkbox"/>	430
11888	13 59:34.473	User Descope LM	Search Court: OK	61	<input checked="" type="checkbox"/>	408
11889	13 59:34.475	User Descope LM	Search Court: Accepted	108	<input checked="" type="checkbox"/>	430
11890	13 59:34.442	User Descope LM	Search Court: Accepted	25	<input checked="" type="checkbox"/>	408
11891	13 59:34.456	User Descope LM	Search List: Manager	80	<input checked="" type="checkbox"/>	430
11892	13 59:34.457	User Descope LM	Search List: Manager	121	<input checked="" type="checkbox"/>	430
11893	13 59:34.473	User Descope LM	Search List: Manager	155	<input checked="" type="checkbox"/>	430

Illustration 8: Results when server switched to other Jetty. Fields with unselected check box represent failed operations.

At this point, error log file got couple of messages similar to this one:

```

ERROR - [2010-03-07T13:59:16.122+0300] - com.avaintech.cab-
rakan.connectivity.impl.DefaultHttpConnectionHandler -
17244592@qtp-17933220-480 - "Got error HTTP status code
[503]"
ERROR - [2010-03-07T13:59:16.128+0300] - com.avaintech.cab-
rakan.http.impl.DefaultSearchHandler - 17244592@qtp-17933220-
480 - "Caught exception while handling request."
java.lang.Exception: Got error HTTP status code [503]
FATAL - [2010-03-07T13:59:17.766+0300] - "Writing event to
closed stream."

```

Some of the queued instances failed because Jetty hasn't startup X-Archive backend before Queue. The others were processed successfully.

On the other hand, instances that were under transaction had the following action log:

REGULAR - 0 - put - Error - Action failed with message [Internal error.].

ROLLBACK - 0 - put - Error - Got Bad Request from critical action handler, rollback failed.¢

4.5 Results of stress-testing X – WFM on Windows

All the servers were set up, and the Jmeter test was initiated. Then after approximately 20 minutes (just when the test reaches peak performance), Server 3 was shut down, so the Server 4 took its place. After another 10 minutes of testing, the Jmeter test was stopped, and both servers shutdown.

From the log-files it was clear, that forms which were “in process” during Servers 3 shutdown got stuck in that phase, and rollback for them (i.e. reverting the modifications related to the instance in question) failed. On the other hand, most of the forms that were only queueing were processed. Some, however, failed since Jetty first started up Queue and only then X-Archive backend.

Altogether, 1054 instances were sent, 1032 of them got processed successfully, 8 failed to be processed at all due to temporary unavailability of the service, 9 were stuck in half-processed state and unreachable through user interface, and 5 were successfully rolled back when main server was started up.

4.6 Result assertion

As it was noted before, in order to be able to deal with high load, X – WFM has a queue mechanism, which stores the basic information about each request (like request ID, instance ID to be processed, its status and instance itself as a blob) on disk, and feeds them to the core part of X – WFM one by one. That help to avoid excessive memory consumption by the application in case of a big number of requests to be processed.

Also, each X–WFM servlet that deals with processing received information has certain algorithms that run on its startup and shut down. The startup ones usually instantiate all necessary processing tools and objects and also rollback any instances they find in temp storage (in case of queue module, for example). Whereas shut down algorithms stops all the processing, saves information to transaction log (so it can be processed when system starts up again), and cleans up all what was created during startup.

It seems that shut down operation didn't had time enough to properly cancel all the processing instances, neither to store or update information about them properly. This resulted in several instances being stuck in a half-digested form, when some part of metadata information regarding instance was updated, while the other part wasn't. As a result, on startup queue mechanism first tried to continue process those instances step by step, but since they were already modified, queue wasn't able to find them and failed the whole transaction. Rolling back to original state also didn't worked out exactly for the same reason. Another problem was incorrect applications boot-up order, which caused several instances to fail with the same “service unavailable” error.

Rollback of instances that were only queued was completed successfully. It also makes sense that they were rolled back and not processed – it is quite likely that users, finding out that their form didn't went through, will try to submit form again. Regardless of which server will be active at the moment, if the original queued instance also will be processed, we'll end up with forms containing duplicate information, which is not very good.

Except for that problem, everything worked just as usual.

On the other hand, this problem was also found later on, when I've tried the same test but in a one server setup – the instances that were in process at the time were broken.

4.7 Possible solutions

One possible way to deal with that problem would be to refine the way queue processes instances on startup. It would be logical to rollback all instances that were in “processing” stage, since the end user most likely received a “server error” message when submitting a form. And, as we can see from the log, this is what exactly queue tries to do on the startup; however since it tries to rollback instances in a usual way, with user rights validation etc., it fails. Another problem is that queue cannot locate the current instance – there are more than one parameter used when searching for it, and for example in case of update, some parameters might have been changed, but due shut down weren't written to transaction log.

This problem then should be addressed in a bit different way than usual rollback – the system should ignore user rights and just rollback the instance to it's original state. Another problem here is that currently queue doesn't store that original state – only steps that were taken during transaction operation. It also rolls back the instance to original state step by step – hence the first step usually fails for the reasons mentioned above.

Thus, in this special case the system should probably ignore all the steps that were taken before, and simply change all the information related to the particular instance id to information stored as “original state”. Alternatively, since all the changes are stored on a database level in a history node tree, certain node can be marked as starting point for each transaction, so during rollback on startup operation system would rollback the instance to the latest starting point history node.

Shared partition for queue temporary files doesn't really matters in this case – queue of the fail-over server will act the same way as queue of main server after restart. Besides, as it was mentioned before, one-server installation has exactly the same problem.

However, how likely is this problem to occur? During the test, system (at its peak) consumed roughly 1500 megabytes of operating memory, which leaves it with roughly 600 megabytes of memory still available for utilization. Further more, since java virtual machine has a limit of memory it can utilize (2 gigabytes total), the application can be split into several Jetty's, each running certain module (up to eleven Jetty's, five per each cocoon module and front-end javascripts, and six per each Java module). Thus they can utilize enormous (in terms of 2010) amount of memory, and each module becomes very unlikely to crash due to out of memory. However, here another problem arises: failure of

one of the Jetty's will still make application look like its working fine, when it is actually not.

Nevertheless, so far, none of the customers using the system reported that system crashed due to lack of memory, hence it can be assumed that the system is rather thread-safe. Thus, any other problems that might occur (collapse of a network, hardware malfunction), as long as they don't cause Jetty to shut down, won't result in any data loss, as our worst scenario. Even though the users won't be able to access the system, it will still be able to process all the data it received. Then again, our application can't really control hardware or network problems.

5 Summary and conclusion

X – WFM does perform as good in failover clustering system as in system with one server only. As it was shown before, there is a possibility of data loss during a sudden shut down of a Jetty web server which runs application. This problem, however, is unrelated to clustering environment.

After this problem will be addressed, system can be used in a active/passive clustering environment in case both systems will have shared database.

As for the active/active clustering environment, additional testing required in case of need for support of such architectures. Then again, X –WFM performs fast enough on one server as well, and so far there was no need for faster response time.

6 Bibliography

1. Windows HPC 2008 Product documentation (<http://www.microsoft.com/hpc/en/us/product-documentation.aspx> – accessed 15.11.2009) – Official documentation for Microsoft Cluster Server 2008. Would be useful to compare both 2003 and 2008 versions, whether there are big differences between them.
2. Technical reference for Microsoft Windows Compute Cluster Server 2003 ([http://technet.microsoft.com/en-us/library/cc720095\(WS.10\).aspx](http://technet.microsoft.com/en-us/library/cc720095(WS.10).aspx) – accessed 16.11.2009) – Official docs for MSCS 2003
3. Enrique Vargas. High Availability Fundamentals, Sun Blueprints Online, November 2000 (<http://www.sun.com/blueprints/1100/HAFund.pdf> – accessed 19.04.2010) – Article about different system categories aimed to help find a server type best suited to specific business availability needs.
4. IBM iSeries Information Center, Version 5 Release 3 (<http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp> – accessed 18.04.2010) – Documentation related to IBM's iSeries servers. Also contains quite a few general technical descriptions regarding networking principles and technologies
5. Basic structure of Linux-HA (<http://www.linux-ha.com/BasicArchitecture> – accessed 27.10.2009) – Basic structure of Linux-HA solution nicely explained. The same site also contains guides on installing and running the system, as well as examples of systems that use Linux-HA as their cluster solution
6. X – WFM 3 documentation
7. Candace Leiden, Marshall Wilensky. TCP/IP for dummies, 6th edition. August 2009 – Chapter IV has a rather ascetic yet easy to understand information about clustered resources and high availability
8. HP-UX System Administrator's Guide – A guide dedicated to managing Hewlett-Packard's popular UNIX operating system HP-UX. Apart from other things, it also has some information on how to configure and manage cluster with this system, as well as some general information on clustering. (<http://docstore.mik.ua/manuals/hp-ux/en/5992-4580/index.html> – accessed 11.11.2009)
9. Apache JMeter user guide (<http://jakarta.apache.org/jmeter/usermanual/> - Accessed 12.01.2010)